Federated PCA with Adaptive Rank Estimation

Andreas Grammenos[1,3]*, Rodrigo Mendoza-Smith[2], Cecilia Mascolo[1], Jon Crowcroft[1,3]

[1]Computer Lab, University of Cambridge
[2]Engineering Department, University of Oxford
[3]Alan Turing Institute

Abstract

In many online machine learning and data science tasks such as data summarisation and feature compression, d-dimensional vectors are usually distributed across a large number of clients in a decentralised network and collected in a streaming fashion. This is increasingly common in modern applications due to the sheer volume of data generated and the clients' constrained resources. In this setting, some clients are required to compute an update to a centralised target model independently using local data while other clients aggregate these updates with a low-complexity merging algorithm. However, some clients with limited storage might not be able to store all of the data samples if d is large, nor compute procedures requiring at least $\Omega(d^2)$ storage-complexity such as Principal Component Analysis, Subspace Tracking, or general Feature Correlation. In this work, we present a novel federated algorithm for PCA that is able to adaptively estimate the rank r of the dataset and compute its r leading principal components when only $O(dr)$ memory is available. This inherent adaptability implies that r does not have to be supplied as a fixed hyper-parameter which is beneficial when the underlying data distribution is not known in advance, such as in a streaming setting. Numerical simulations show that, while using limited-memory, our algorithm exhibits state-of-the-art performance that closely matches or outperforms traditional non-federated algorithms, and in the absence of communication latency, it exhibits attractive horizontal scalability.

1 Introduction

In recent years, the advent of edge computing in smartphones, IoT and cryptocurrencies has induced a paradigm shift in distributed model training and large-scale data analysis. Under this new paradigm, data is generated by commodity devices with hardware limitations and severe restrictions on data-sharing and communication, which makes the centralisation of the data extremely difficult. This has brought new computational challenges since algorithms do not only have to deal with the sheer volume of data generated by networks of devices, but also leverage the algorithm's voracity, accuracy, and complexity with constraints on hardware capacity, data access, and device-device communication. In light of this, the necessity of being able to analyse large-scale decentralised datasets and extract useful insights out of them is becoming more prevalent than ever before. Seminal work in this domain has been made, but mainly in the context of deep neural networks, see McMahan et al. (2016); Konečnỳ et al. (2016). Specifically, in Konečnỳ et al. (2016) a *federated* method for training of neural networks was proposed. In this setting one assumes that each of a large number of independent *clients* can contribute to the training of a centralised model by computing local updates with their own data and sending them to the client holding the centralised model for aggregation. Ever since the publication of this seminal work, interest in federated algorithms for training neural networks has surged, see Smith et al. (2017), He et al. (2018), Geyer et al. (2017). Despite of this, federated adaptations of classical

*Correspondence to: Andreas Grammenos <ag926@cl.cam.ac.uk>

Preprint. Under review.

data analysis techniques are still missing. Out of the many techniques available, Principal Component Analysis (PCA) Pearson (1901); Jolliffe (2011) is arguably the most ubiquitous one for discovering linear structure or reducing dimensionality in data, so has become an essential component in inference, machine-learning, and data-science pipelines. In a nutshell, given a matrix $\mathbf{Y} \in \mathbb{R}^{d\times n}$ of n feature vectors of dimension d, PCA aims to build a low-dimensional subspace of $\mathbb{R}^d$ that captures the directions of maximum variance in the data contained in $\mathbf{Y}$. Apart from being a fundamental tool for data analysis, PCA is often used to reduce the dimensionality of the data to minimise the cost of computationally expensive operations. For instance, before applying t-SNE Maaten and Hinton (2008) or UMAP McInnes et al. (2018). Hence, a federated algorithm for PCA is not only desired when data-ownership is sought to be preserved, but also from a computational viewpoint.

Herein, we propose a federated algorithm for PCA. The computation of PCA is related to the Singular Value Decomposition (SVD) Eckart and Young (1936); Mirsky (1966) which can decompose any matrix into a linear combination of orthonormal rank-1 matrices weighted by positive scalars. In the context of high-dimensional data, the main limitation stems from the fact that, in the absence of structure, performing PCA on a matrix $\mathbf{Y} \in \mathbb{R}^{d\times n}$ requires $O(d^2n + d^3)$ computation time and $O(d^2)$ memory. This cubic computational complexity and quadratic storage dependency on d makes the cost of PCA computation prohibitive for high-dimensional data, though it can often be circumvented when the data is sparse or has other type of exploitable structure. Moreover, in some decentralised applications, the computation has to be done in commodity devices with $O(d)$ storage capabilities, so a PCA algorithm with $O(d)$ memory dependency is highly desirable. On this front, there have been numerous recent works in the streaming setting that try to tackle this problem, see Mitliagkas et al. (2014, 2013); Marinov et al. (2018); Arora et al. (2012, 2016); Boutsidis et al. (2015). However, most of these methods do not naturally scale well nor can they be parallelised efficiently despite their widespread use, e.g. Bouwmans and Zahzah (2014); Boutsidis et al. (2015). To overcome these issues a reliable and federated scheme for large decentralised datasets is highly desirable. In this work, we exploit a known paradigm from the randomised linear algebra literature dubbed *sketch and solve* Woodruff et al. (2014). This paradigm aims to save computational costs by producing a *summary*, or reduced representation of the data, to approximately solve an instance of the original problem. In matrix computation terms, this paradigm is particularly suitable if summaries can be computed incrementally as the data is discovered, such as in the streaming setting.

Summary of contributions: Our main contribution is a modular, federated algorithm for PCA designed for the combined setting of stochastic and streaming data. Our algorithm is comprised out of two distinct and independent components: (1) An algorithm for the incremental, decentralised computation of local updates to PCA, (2) a low-complexity merging procedure to aggregate these incremental updates together. The incremental component is based on a scheme for streaming linear dimensionality reduction proposed in Eftekhari et al. (2018) which belongs to the family of deterministic sketching algorithms in the sequence Ghashami et al. (2016); Liberty (2013); Ghashami and Phillips (2014). However, contrary to previous algorithms, our algorithm is able to adaptively estimate the rank r and adjust the number of principal components by controlling the contribution of the least significant singular value to the total variance. The aggregation component in the streaming setting is the result of a corollary of SVD (Lemma 1), which leads to an improved version of the algorithm proposed in Rehurek (2011). Moreover, our algorithm is designed to work under the assumption that we are only allowed to do *one pass* through each column of $\mathbf{Y}$ using an $O(d)$-memory device.

2 Notation & Preliminaries

This section introduces the notational conventions used throughout the paper. For integers $m \leq n$ we use the shorthand $[m, n] = \{m, \cdots, n\}$ and $[n]$ for the special case $m = 1$. We use lowercase letters y for scalars, bold lowercase letters $\mathbf{y}$ for vectors, bold capitals $\mathbf{Y}$ for matrices, and calligraphic capitals $\mathcal{Y}$ for subspaces. If $\mathbf{Y} \in \mathbb{R}^{d\times n}$ and $S \subset [m]$, then $\mathbf{Y}_S$ is the block composed of columns indexed by S. In particular, when $S = [k]$ for some $k \in \mathbb{N}$ we write $\mathbf{Y}_{[k]}$. We reserve $\mathbf{0}_{m\times n}$ for the zero matrix in $\mathbb{R}^{m\times n}$ and $\mathbf{I}_n$ for the identity matrix in $\mathbb{R}^{n\times n}$. Additionally, we use $\|\cdot\|_{\mathrm{F}}$ to denote the Frobenius norm operator and $\|\cdot\|$ to denote the ℓ_2 norm. If $\mathbf{Y} \in \mathbb{R}^{d\times n}$ we let $\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ be its full SVD formed from unitary $\mathbf{U} \in \mathbb{R}^{d\times d}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ and diagonal $\mathbf{\Sigma} \in \mathbb{R}^{d\times n}$ having $\mathbf{\Sigma}_{i,i} = \sigma_i(\mathbf{Y}) \geq 0$. The values $\sigma_1(\mathbf{Y}) \geq \cdots \geq \sigma_k(\mathbf{Y})$ are the singular values of $\mathbf{Y}$. If $1 \leq r \leq \min(m, n)$, we let $\rho_r^2(\mathbf{Y}) = \sum_{i\geq r+1} \sigma_i^2(\mathbf{Y})$ be the *residual* of $\mathbf{Y} \in \mathbb{R}^{m\times n}$ and $\mathrm{SVD}_r(\mathbf{Y}) = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^T$ be its *best*

rank-r approximation, which is the solution to

$$\inf_{\mathbf{Z}\in\mathbb{R}^{d\times n}} \|\mathbf{Z}-\mathbf{Y}\|_F \text{ subject to } \operatorname{rank}(\mathbf{Z}) \leq r.$$

We will often abuse notation and write $\mathrm{SVD}_r(\mathbf{Y})$ to denote the triplet $[\mathbf{U}_r, \mathbf{\Sigma}_r, \mathbf{V}_r]$. Both uses of SVD_r should be clear from the context. Finally, we let $[\mathbf{Q}, \mathbf{R}] = \mathrm{QR}(\mathbf{Y})$ be the QR factorisation of $\mathbf{Y}$ and $\lambda_1(\mathbf{Y}) \geq \cdots \geq \lambda_k(\mathbf{Y})$ be its eigenvalues when $d = n$.

Streaming Model: A data stream is defined as a feature vector sequence $\mathbf{y}_{t_0}, \mathbf{y}_{t_1}, \mathbf{y}_{t_2}, \ldots$ with the property that $t_{i+1} - t_i > 0$ for any $i \in \mathbb{N}$. In this work, we shall assume that data streams are vectors in $\mathbb{R}^d$ indexed by the natural numbers, which at any time $n \in \mathbb{N}$ can be arranged in a matrix $\mathbf{Y} \in \mathbb{R}^{d\times n}$.

Federated learning: Federated Learning Konečnỳ et al. (2016) is a machine-learning paradigm that considers how a large number of *clients* owning different data-points can contribute to the training of a *centralised model* by locally computing updates with their own data and merging them to the centralised model without sharing data between each other. Our method resembles the distributed agglomerative summary model (DASM) Tanenbaum and Van Steen (2007) in which updates are aggregated in a "bottom-up" approach following a tree-structure. That is, by arranging the nodes in a tree-like hierarchy in such a way that for any sub-tree, the leaves compute intermediate results and propagate them to the roots for merging or summarisation. In our model, the summaries only have to travel upwards only once to be combined. This permits minimal synchronisation between the nodes so, for the purposes of this work, we do not model any issues related to synchronisation.

3 Federated PCA

We consider a decentralised dataset $D = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \subset \mathbb{R}^d$ scattered across M clients. The dataset D can be stored in a matrix $\mathbf{Y} = \left[\mathbf{Y}^1 | \mathbf{Y}^2 | \cdots | \mathbf{Y}^M\right] \in \mathbb{R}^{d\times n}$ with $n \gg d$ and such that $\mathbf{Y}^i \in \mathbb{R}^{d\times n_i}$ is owned by client $i \in [M]$. We assume that each $\mathbf{Y}^i$ is generated in a streaming fashion and that due to resource limitations it cannot be stored in full. Furthermore, under the DASM we assume that the M clients in the network can be arranged in a tree-like structure with $q > 1$ levels and approximately $\ell > 1$ leaves per node. Without loss of generality, in this paper we assume that $M = \ell^q$. Our federated algorithm for PCA is given in Algorithm 1.

Algorithm 1: Federated PCA

Data: $\mathbf{Y} = \left[\mathbf{Y}^1 | \cdots | \mathbf{Y}^M\right] \in \mathbb{R}^{d\times n}$ such that $\mathbf{Y}^i \in \mathbb{R}^{d\times n_i}$ belongs to client $i \in [M]$;
$r \in [d]$, rank estimate; k, number of subspace aggregations per batch;
α, β, bounds on $\sigma_{\tau,r}$ if local updates are computed with SAPCA;
Result: $(\mathbf{U}', \mathbf{\Sigma}') \in \mathbb{R}^{d\times r} \times \mathbb{R}^{r\times r}$ such that $\mathbf{U}' \approx \mathbf{U}_{[r]}$ and $\mathbf{\Sigma}' \approx \mathbf{\Sigma}_{[r]}$ with $\mathrm{SVD}(\mathbf{Y}) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$;
$\mathbf{Y}^{1,i} \leftarrow \mathbf{Y}^i$ for $i \in [M]$;
for *each level* $p \in [q]$ **do**
  // Phase I: Estimate local updates
  for *each client* $i \in [M/\ell^{p-1}]$ *in level* p **do**
    $[\mathbf{U}^{p,i}, \mathbf{\Sigma}^{p,i}, \sim] \leftarrow \mathrm{SVD}_r(\mathbf{Y}^{p,i})$
  end
  // Phase II: Merging estimations
  Obtain $(\mathbf{U}^{p+1,i}, \mathbf{\Sigma}^{p+1,i})$ by merging $\left\{\left(\mathbf{U}^{p,i}, \mathbf{\Sigma}^{p,i}\right)\right\}_{i\in[M/\ell^{p-1}]}$ recursively on batches of size k;
end
$(\mathbf{U}', \mathbf{\Sigma}') \leftarrow \left(\mathbf{U}^{q+1,1}_{[r]}, \mathbf{\Sigma}^{q+1,1}_{[r]}\right)$;

Note that Algorithm 1, invokes two separate sets of procedures corresponding to the computation of local updates and the merging of resulting sub-spaces. In the inner-most loop, each client $i \in [M/\ell^{p-1}]$ collects or generates $\mathbf{Y}^{p,i}$ in a streaming fashion and computes an estimate of $\mathrm{SVD}_r(\mathbf{Y}^{p,i})$ using SPCA or SAPCA (Algorithm 3). Then, the estimates for level p are aggregated by the calling `merge` (Algorithm 2) recursively on subspace batches of size k. Due to the recursive nature of the procedure, only $M/\ell^p \log(M/\ell^p) = \ell^{q-p}\log(\ell^{q-p})$ merging operations are required at level p. Detailed descriptions of Algorithms 2-3 are given in the following sections.

3.1 Merging

Our algorithmic constructions are built upon the concept of *subspace merging* in which two subspaces $\mathcal{S}_1 = (\mathbf{U}_1, \mathbf{\Sigma}_1) \in \mathbb{R}^{r_1 \times d} \times \mathbb{R}^{r_1 \times r_1}$ and $\mathcal{S}_2 = (\mathbf{U}_2, \mathbf{\Sigma_2}) \in \mathbb{R}^{r_2 \times d} \times \mathbb{R}^{r_2 \times r_2}$ are *merged* together to produce a subspace $\mathcal{S} = (\mathbf{U}, \mathbf{\Sigma}) \in \mathbb{R}^{r \times d} \times \mathbb{R}^{r \times r}$ describing the combined r principal directions of $\mathcal{S}_1$ and $\mathcal{S}_2$. One can merge two sub-spaces by computing a truncated SVD on a concatenation of their bases. Namely,

$$[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V^T}] \leftarrow \mathrm{SVD}_r([\lambda \mathbf{U}_1 \mathbf{\Sigma}_1, \mathbf{U}_2 \mathbf{\Sigma}_2]), \tag{1}$$

where $\lambda \in (0, 1]$ a *forgetting factor* that allocates less weight to the previous subspace $\mathbf{U}_1$. An efficient version of (1) is presented in Algorithm 4 (Appendix). In Rehurek (2011), it is shown how (1) can be further improved when $\mathbf{V}^T$ is not required and we have knowledge that $\mathbf{U}_1$ and $\mathbf{U}_2$ are already orthonormal. This is done by building a basis $\mathbf{U}'$ for $\mathrm{span}((\mathbf{I} - \mathbf{U_1}{\mathbf{U_1}}^T)\mathbf{U_2})$ via the QR factorisation and then computing the SVD decomposition of a matrix $\mathbf{X}$ such that

$$[\mathbf{U_1}\mathbf{\Sigma_1}, \mathbf{U_2}\mathbf{\Sigma_2}] = [\mathbf{U_1}, \mathbf{U}']\mathbf{X} \tag{2}$$

It is shown in (Rehurek, 2011, Chapter 3) that this yields an $\mathbf{X}$ of the form

$$\mathbf{X} = \begin{bmatrix} \mathbf{U_1^T U_1 \Sigma_1} & \mathbf{U_1^T U_2 \Sigma_2} \\ \mathbf{U'}^T\mathbf{U_1} & \mathbf{U'^T U_2 \Sigma_2} \end{bmatrix} = \begin{bmatrix} \mathbf{\Sigma_1} & \mathbf{U_1^T U_2 \Sigma_2} \\ 0 & \mathbf{R_p \Sigma_2} \end{bmatrix} \tag{3}$$

where $[\mathbf{U}, \mathbf{R}_p] = \mathrm{QR}((\mathbf{I} - \mathbf{U}_1\mathbf{U}_1^T)\mathbf{U}_2)$. This procedure is shown in Algorithm 2.

Algorithm 2: MergeSubspaces (Fastest subspace-merging algorithm)

Data: $(\mathbf{U}_1, \mathbf{\Sigma}_1) \in \mathbb{R}^{d \times r_1} \times \mathbb{R}^{r_1 \times r_1}$, $(\mathbf{U}_2, \mathbf{\Sigma}_2) \in \mathbb{R}^{d \times r_2} \times \mathbb{R}^{r_2 \times r_2}$, subspaces to be merged;
$r \in [r]$, rank estimate ; $\lambda_1 \in (0, 1)$, forgetting factor for $\mathbf{U}_1$; $\lambda_2 \geq 1$, enhancing factor for $\mathbf{U}_2$;
Result: $(\mathbf{U}', \mathbf{\Sigma}') \in \mathbb{R}^{d \times r} \times \mathbb{R}^{r \times r}$ merged subspace;
$\mathbf{Z}_r \leftarrow \mathbf{U}_1^T\mathbf{U}_2$;
$[\mathbf{U}_p, \mathbf{R}_p] \leftarrow \mathrm{QR}(\mathbf{U}_2 - \mathbf{U}_1\mathbf{Z}_r)$;
$[\mathbf{U}_R, \mathbf{\Sigma}', \sim] \leftarrow \mathrm{SVD}_r\left(\begin{bmatrix} \lambda_1\mathbf{\Sigma}_1 & \mathbf{Z}_r\mathbf{\Sigma}_2 \\ 0 & \lambda_2\mathbf{R}_p\mathbf{\Sigma}_2 \end{bmatrix}\right)$;
$\mathbf{U}' \leftarrow [\mathbf{U}_1, \mathbf{U}_p]\mathbf{U}_R$;

The merging procedure outlined above can be generalised to multiple subspaces. A proof of this was given by Iwen and Ong (2016). In Lemma 1 we extend this result to the case of streaming data. The proof is delayed to the Appendix.

Lemma 1 (Streaming partial SVD uniqueness)**.** *Let* $\mathbf{Y} = \left[\mathbf{Y}^1|\mathbf{Y}^2|\cdots|\mathbf{Y}^M\right] \in \mathbb{R}^{d \times n}$ *with* $n \gg d$ *and* $\mathbf{Y}^i \in \mathbb{R}^{d \times n_i}$ *with* SVD *given by* $\mathbf{Y}^i = \hat{\mathbf{U}}^i\hat{\mathbf{\Sigma}}^i(\hat{\mathbf{V}}^i)^T$*. Assume streaming data and that at any given time only only* $b \ll n_i$ *vectors can be stored in each client. Let* $\mathbf{Z} := [\hat{\mathbf{U}}^1\hat{\mathbf{\Sigma}}^1|\cdots|\hat{\mathbf{U}}^M\hat{\mathbf{\Sigma}}^M]$*, and let* $\mathbf{Y} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T$ *and* $\mathbf{Z} = \hat{\mathbf{U}}'\hat{\mathbf{\Sigma}}'(\hat{\mathbf{V}}')^T$ *be the reduced* SVD *decompositions of* $\mathbf{Y}$ *and* $\mathbf{Z}$*. Then* $\hat{\mathbf{\Sigma}} = \hat{\mathbf{\Sigma}}'$*, and* $\hat{\mathbf{U}} = \hat{\mathbf{U}}'\mathbf{B}$*, where* $\mathbf{B}$ *is a unitary block diagonal matrix. If none of the nonzero singular values are repeated then* $\mathbf{B} = \mathbf{I}$*.*

We stress that result proved in Iwen and Ong (2016) is incremental, but *not* streaming. This means that every result has to be computed in-full in order to be processed, merged, and propagated for the user to get a final result, so is not fit for a federated computing approach.

3.2 Estimation of local updates: Streaming, adaptive PCA

In this section we introduce SPCA and SAPCA which are the streaming algorithms clients use to compute local updates to the centralised model. Consider a sequence $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \subset \mathbb{R}^d$ of feature vectors and let their concatenation at time $\tau \leq n$ be

$$\mathbf{Y}_{[\tau]} = [\ \mathbf{y}_1 \quad \mathbf{y}_2 \quad \cdots \quad \mathbf{y}_\tau\ ] \in \mathbb{R}^{d \times \tau}. \tag{4}$$

A block of size $b \in \mathbb{N}$ is formed by taking b contiguous columns of $\mathbf{Y}_{[\tau]}$. Hence, a matrix $\mathbf{Y}_{[\tau]}$ with $r \leq b \leq \tau$ induces $K = \lceil \tau/b \rceil$ blocks. For convenience, we assume $K \in \mathbb{N}$, so that $\tau = Kb \in \mathbb{N}$. In this case, block $k \in [K]$ corresponds to the sub-matrix containing columns $S_k = [(k-1)b + 1, kb]$.

It is assumed that all blocks S_k are owned and observed exclusively by client $i \in [M]$, but that due to resource limitations it can not store them all. Hence, once client i has observed $\mathbf{Y}_{S_k} \in \mathbb{R}^{d\times b}$ it uses it to update its estimate $\widehat{\mathbf{Y}}_{[(k-1)b],r}$ of the r principal components of $\mathbf{Y}_{[kb]}$ and then releases $\mathbf{Y}_{S_k}$ from memory. If $\widehat{\mathbf{Y}}_{0,r}$ is the empty matrix, the r principal components of $\mathbf{Y}_{[\tau]}$ can be estimated by

$$\widehat{\mathbf{Y}}_{[kb],r} = \mathrm{SVD}_r\left(\begin{bmatrix} \widehat{\mathbf{Y}}_{[(k-1)b],r} & \mathbf{Y}_{S_k} \end{bmatrix}\right) \in \mathbb{R}^{d\times kb}. \tag{5}$$

This algorithm is called MOSES and was proposed in Eftekhari et al. (2018). Its output after K iterations is $\widehat{\mathbf{Y}}_{[Kb],r} = \mathrm{SVD}_r(\mathbf{Y}_{[\tau]})$, which, as mentioned above, contains both an estimate of leading r principal components of $\mathbf{Y}_{[\tau]}$ and the projection of $\mathbf{Y}_{[\tau]}$ onto this estimate. The local subspace estimation of our federated algorithm is inspired on an implementation of iteration (5) presented in Eftekhari et al. (2018). However, our algorithm is designed to achieve substantial computational savings by only tracking the left principal subspace and, contrary to other algorithms, the singular value estimates. Additionally, the nodes in our algorithm can adjust, independently of each other, their rank estimate based on the distribution of the data seen so far. This is convenient because we expect to have nodes which observe different distributions and will likely require to adjust the number of principal components kept in order to accurately track the data distribution over time. While *energy thresholding methods* Al-Kandari and Jolliffe (2005); Papadimitriou et al. (2005) are a natural and established way to achieved this, these methods only work well when updates are performed for each feature vector. Moreover, according to our experiments, they also under-perform with block-based approaches. To this end, we propose a simple and efficient regularisation scheme based solely on the estimate of the current singular values and their contribution to the total approximation discovered so far. Specifically, letting $\sigma_{\tau,i}$ be the i-th singular value of the dataset at time $\tau > 0$, we use $\sigma_{\tau,r}$ to control the *lower bound* on the total variance $\sum_{i=1}^{r} \sigma_{\tau,i}$. We do this by letting $\alpha, \beta > 0$ be minimum and maximum contributions to the variance of the dataset and enforcing

$$C_{\tau,r} = \frac{\sigma_{\tau,r}}{\sum_{i=1}^{r} \sigma_{\tau,i}} \in [\alpha, \beta]. \tag{6}$$

That is, by increasing r whenever $C_{\tau,r} > \beta$ and decreasing it when $C_{\tau,r} < \alpha$. This adjustment happens only once per block in order to prevent the addition of too many principal components in one go, but point out that a number of variations to this strategy are possible. For example, a user could also implement a *hold-off* duration to prevent updates in r for a predefined number of blocks. Our algorithm is presented in Algorithm 3. We use the labels SPCA and SAPCA to distinguish the variant with adaptive rank estimation.

Note that Algorithm 3 can, if desired, explicitly maintain the estimates of principal components along with the projected data. Moreover, its storage and computational requirements are nearly optimal for the given objective since, at iteration k, it only requires $O(r(d+kr))$ bits of memory and $O(r^2(d+kb)) = O(r^2(d+kr))$ flops. We point out that SPCA inherits some theoretical guarantees which ensure that its output differs from the offline SVD in only a small polynomial factor, see Appendix B. Finally, in Lemma 2 we provide a simple bound on the error of SAPCA together with an guarantee on time-order independence in the data, which is essential in a decentralised, federated system to guarantee robustness. The proofs to these statements are given in the Appendix.

Lemma 2 (SPCA / SAPCA properties)**.** *Let* $\mathbf{Y} \in \mathbb{R}^{d\times n}$*,* $r \in [d]$*, and* $\alpha, \beta > 0$*. Then,*

1. $\|\mathbf{Y} - \hat{\mathbf{Y}}\|_F \leq \rho_{r_{\min}}(\mathbf{Y})$ *where* $\hat{\mathbf{Y}} = \mathrm{SAPCA}(\mathbf{Y}, r, \alpha, \beta)$ *and* $r_{\min} = r_{\min}(\alpha, \beta)$ *is the minimum rank estimated by* SAPCA.
2. *If* $\mathbf{P} \in \mathbb{R}^{n\times n}$ *is a row permutation of the identity, then* $\mathrm{SPCA}(\mathbf{Y}) = \mathrm{SPCA}(\mathbf{YP})$*. A similar result holds for* SAPCA.

4 Experimental Evaluation

To validate our scheme, we evaluate the empirical performance of SAPCA by comparing it against competing algorithms using synthetic and real datasets[2]. All our experiments were computed on a server using Dual Intel Xeon X5650 CPUs with 12 cores at 2.66GHz, 32 GB 1333 MHz ECC

[2]To foster reproducibility both code and datasets used are made publicly available here: `https://github.com/andylamp/federated_pca`.

Algorithm 3: Streaming PCA / Streaming Adaptive PCA (SPCA/SAPCA)

Data: Feature vectors $\{\mathbf{y}_t\}_{t\in[\tau]} \in \mathbb{R}^d$, with $\tau = Kb$;
$r \in [d]$, rank estimate ; $b \in [r, \infty)$ the desired block size;
α, β, bounds on $\sigma_{\tau,r}$ (If algorithm is SPCA);
Result: $\{(\widehat{\mathbf{U}}_{kb,r}, \widehat{\mathbf{\Sigma}}_{kb,r}) : k \in [K]\}$, estimations of the principal subspaces for the k-th block.
for $k \in [K]$ **do**
  $\mathbf{Y}_{S_k} \leftarrow \left[\mathbf{y}_{(k-1)b+1} \cdots \mathbf{y}_{kb}\right]$, // $S_k = [(k-1)b+1, kb]$;
  if $k = 1$ **then**
    $[\widehat{\mathbf{U}}_{b,r}, \widehat{\mathbf{\Sigma}}_{b,r}, \sim] \leftarrow \mathrm{SVD}_r(\mathbf{Y}_{S_k})$;
  else
    $\mathbf{Z}_k \leftarrow \widehat{\mathbf{U}}^T_{(k-1)b,r}\mathbf{Y}_{S_k} \in \mathbb{R}^{r\times b}$;
    $[\widehat{\mathbf{U}}_k, \mathbf{R}_k] \leftarrow \mathrm{QR}(\mathbf{Y}_k - \widehat{\mathbf{U}}_{(k-1)b,r}\mathbf{Z}_k)$;
    $[\mathbf{U}_p, \widehat{\mathbf{\Sigma}}_{kb,r}, \sim] \leftarrow \mathrm{SVD}_r\left(\begin{bmatrix} \widehat{\mathbf{\Sigma}}_{(k-1)b,r} & \mathbf{Z}_k \\ \mathbf{0}_{b\times r} & \mathbf{R}_k \end{bmatrix}\right)$;
    $\widehat{\mathbf{U}}_{kb,r} \leftarrow [\ \widehat{\mathbf{U}}_{(k-1)b,r} \quad \widehat{\mathbf{U}}_k\ ]\mathbf{U}_p$;
  end
  if *Algorithm is* SAPCA **then**
    if $\sigma_r > \beta\sum_{i=1}^r \sigma_i$ **then**
      $r \leftarrow r + 1$
      $\widehat{\mathbf{U}}_{kb,r} \leftarrow [\widehat{\mathbf{U}}_{kb,r}, e_r]$, $e_r \in \mathbb{R}^d$ is r-th canonical vector.
    else if $\sigma_r < \alpha\sum_{i=1}^r \sigma_i$ **then**
      $r \leftarrow r - 1$
      $\widehat{\mathbf{U}}_{kb,r} \leftarrow (\widehat{\mathbf{U}}_{kb,r})_{[r]}$
    end
  end
end

DDR3 RAM, and Matlab R2019a (build 9.6.0.1099231). The algorithms considered in this instance are, SAPCA, GROUSE Balzano and Wright (2013), Frequent Directions (FD) Desai et al. (2016); Luo et al. (2017), the Power Method (PM) Mitliagkas et al. (2014), and a variant of Projection Approximation Subspace Tracking (PAST) Yang (1995), named SPIRIT (SP) Papadimitriou et al. (2005). As SPCA is derived from MOSES Eftekhari et al. (2018) and returns the same subspace and singular values as MOSES given a *fixed* r; hence, a comparison with MOSES is not done. We assume a dataset like (4) and that for each algorithm alg and each $t \in [\tau]$ an estimate $\widehat{\mathbf{Y}}^{alg} = \mathrm{SVD}_r(\mathbf{Y}_t)$ is computed along with the Frobenius-norm error and Mean Squared Error (MSE). That is,

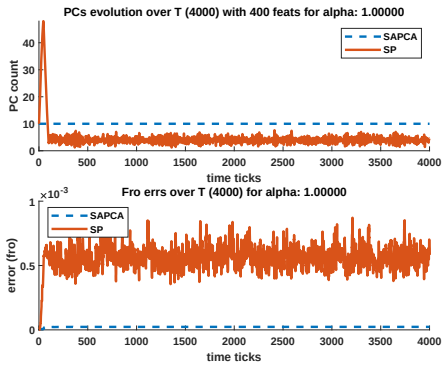
$$\mathrm{Err}^{alg,F}_{t,r} = \|\mathbf{Y}_t - \widehat{\mathbf{Y}}^{alg}_{t,r}\|^2_F, \quad \mathrm{Err}^{alg,\mathrm{mse}}_{t,r} = \frac{1}{t}\|\mathbf{Y}_t - \widehat{\mathbf{Y}}^{alg}_{t,r}\|^2_F, \quad alg \in \{\text{SAPCA, GROUSE, FD, PM, SP}\}.$$

4.1 SAPCA performs favourably compared to other methods in synthetic and real datasets

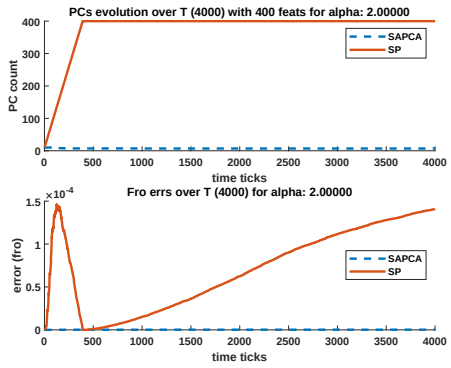
Figures 1a and 1b show the results of our experiments on synthetic data Synth$(\alpha)^{d\times n} \subset \mathbb{R}^{d\times n}$ with $(d, n) = (400, 4000)$ generated independently from a zero-mean multivariate Gaussian with covariance matrix $\mathbf{S}\mathbf{\Lambda}\mathbf{S}^T$, where $\mathbf{S} \in \mathbb{R}^{d\times d}$ is the orthogonalisation of a random Gaussian matrix and $\mathbf{\Lambda} \in \mathbb{R}^{d\times d}$ is a diagonal matrix with $\mathbf{\Lambda}_{i,i} = i^{-\alpha}$ and $\alpha > 0$. In the experiments, we let λ be the forgetting factor of SP. Figure 1a compares SPCA with SP when $(\alpha, \lambda) = (1, 0.9)$ and Figure 1b when $(\alpha, \lambda) = (2, 1)$. While SPCA exhibits relative stability in $\mathrm{Err}^{\mathrm{SPCA},F}_{t,r}$ in both cases, SP exhibits a monotonic increase in the number of principal components estimated when $(\alpha, \lambda) = (2, 1)$. This behaviour is replicated in Figures 1e and 1f where $\mathrm{Err}^{\mathrm{MSE}}$ is computed on SAPCA, SP, and variations of SPCA and PM dubbed SPCA_{lo}, SPCA_{hi}, and PM_{lo} with rank set to the minimum (lo) and maximum (hi) ranks estimated by SAPCA in the simulation. Figures 1c and 1d benchmark SAPCA on real sensor-node *humidity* and *light* datasets[3] obtained from Deshpande et al. (2004) both with dimensionality of $\mathbb{R}^{48\times 7712}$. The figures show that our method performs favourably

[3]Datasets on *volt* and *temperature* readings are also available and used, see Appendix C.1 for detailed description of these datasets and additional experiments. Source of data: `https://www.cs.cmu.edu/afs/cs/project/spirit-1/www/data/Motes.zip`

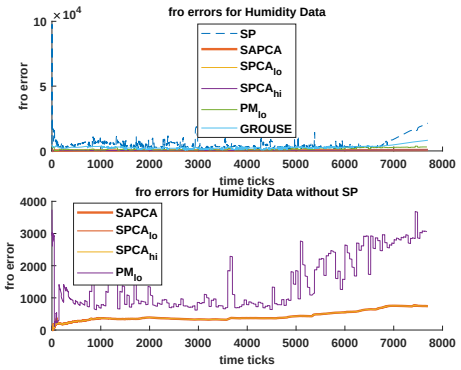
against all competing algorithms. Additional experiments on synthetic and real datasets are given in Appendix C.1.



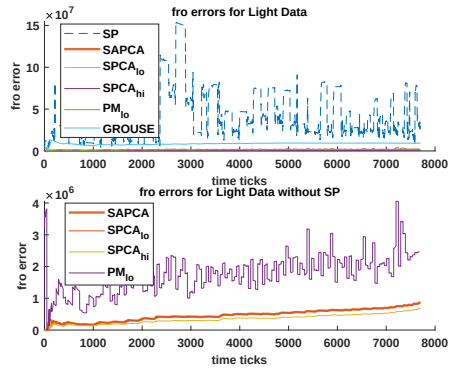
(a) $\|\cdot\|_F$ and PC count on Synth(1) with $\lambda = 0.9$



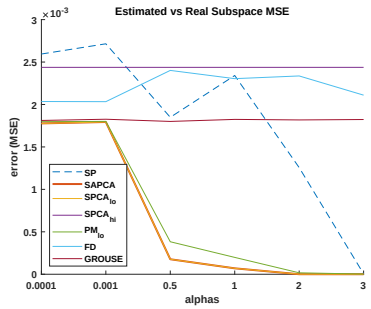
(b) $\|\cdot\|_F$ and PC count on Synth(2) with $\lambda = 1$



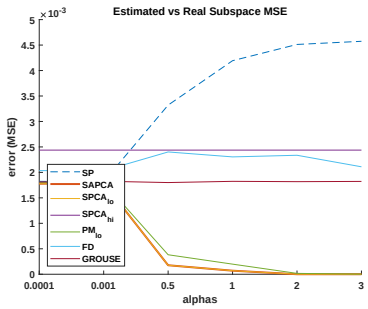
(c) $\|\cdot\|_F$ on *humidity* dataset



(d) $\|\cdot\|_F$ on *light* dataset



(e) Err^{MSE} with $\lambda = 0.9$



(f) Err^{MSE} with $\lambda = 1$

Figure 1: Performance of SAPCA on synthetic and real datasets.

4.2 SAPCA's memory efficiency is considerably better on average than competing methods

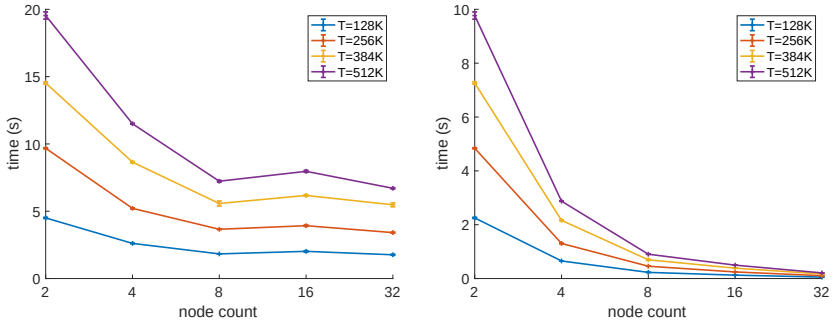
Table 1 reports average and median memory-allocation profiles for each algorithm and each sensor-node dataset provided in Deshpande et al. (2004). The statistics were computed over a sample of ten runs. Table 1 shows that SAPCA's memory efficiency is compares favourably against other algorithms in the experiment. See Appendix C.3 for further details and an extended discussion.

Table 1: Average / median memory allocations

Real Datasets				
Method	Humidity	Light	Voltage	Temperature
SAPCA	**138.11** Kb / **58.99** Kb	**104.00** Kb / **76.03** Kb	204.58 / **23.47** Kb	**187.74** Kb / **113.28** Kb
PM	905.45 Kb / 666.11	685.48 Kb / 685.44 Kb	649.12 Kb / 644.35 Kb	657.57 Kb / 668.27 Kb
GROUSE	2896.61 Kb / 2896.62 Kb	2896.84 Kb / 2896.62 Kb	2772.86 Kb / 2772.62 Kb	3379.62 Kb / 3376.62 Kb
FD	162.70 Kb / 117.92 Kb	170.48 Kb / 127.91 Kb	**114.46** Kb / 112.66 Kb	196.11 Kb / 118.59 Kb
SP	476.68 Kb / 405.01 Kb	1009.03 Kb / 508.11 Kb	348.84 Kb / 351.98 Kb	541.56 Kb / 437.61 Kb

4.3 The federated scheme shows graceful scaling when simulated in a multi-core CPU

To simulate a federated computation environment we compare the average execution times, $\text{time}(\mathbf{Y})$, required to compute PCA on a dataset $\mathbf{Y} \in \mathbb{R}^{d \times T}$. To do this, we fix $d = 1\text{k}$ and for each $T \in \{128\text{k}, 256\text{k}, 384\text{k}, 512\text{k}\}$, report $\frac{1}{|L|}\sum_{\alpha \in L} \text{time}(\mathbf{Y}_\alpha)$ where $\mathbf{Y}_\alpha \sim \text{Synth}(\alpha)^{d \times T}$ and $L = \{10^{i-5} : i \in [5]\} \cup \{2, 3, 4\}$. In Figure 2 we report the time required to compute each sub-problem and merge the results together under different assumptions on the number of computing nodes (maximum of 32 in our simulation). The real time scalings are shown in Figure 2a while the amortised scaling, corresponding to the federated scenario of having one processing core per client, can be seen in Figure 2b. Figure 2a shows that a regression occurs after we exceed the number of available physical cores on our machine (in our case 12) while fixing T to a specific value; this behaviour is normal as, due to the lack of processing nodes, not all of the sub-problems can be executed in concurrently. On the other hand, Figure 2b shows a very graceful scaling in the average execution times in the presence of as many processing nodes as the amount of sub-problems to solve across all values of T. We also assume that each node is independent and completely owns its subset of the (evolving) dataset. More details and discussion can be found in Appendix C.4.



(a) Real (b) Amortised.

Figure 2: SAPCA Performance Scaling for Real and Amortised execution time.

5 Discussion & Conclusions

We introduced a federated streaming algorithm for PCA. Our federated algorithm is the result of two separate innovations. Namely, an adaptive streaming algorithm for computing rank-r approximations (SAPCA), and a fast subspace merging algorithm to aggregate these updates together (MergeSubspaces). We complement our algorithm with several theoretical results that guarantee bounded estimation errors and as well as data permutation invariance. In particular, we extended a result by Iwen and Ong (2016) to guarantee bounded memory in SAPCA, which is crucial to enable horizontal scalability and bring PCA to the federated setting. Our numerical experiments show that SAPCA performs favourably against other methods in terms of convergence, bounded estimation errors, low memory requirements, and also that Federated-PCA scales gracefully when simulated on a multi-core CPU. While our algorithm comes with no guarantees on data-privacy, we point out that it can readily implement the input-perturbation scheme described in Chaudhuri et al. (2013), but we leave this as potential avenue for future work. Another interesting avenue of future work is to devise a computationally efficient scheme for performing PCA in a federated setting but in the presence of missing values.

References

Noriah M Al-Kandari and Ian T Jolliffe. 2005. Variable selection and interpretation in correlation principal components. *Environmetrics: The official journal of the International Environmetrics Society* 16, 6 (2005), 659–672.

Raman Arora, Andrew Cotter, Karen Livescu, and Nathan Srebro. 2012. Stochastic optimization for PCA and PLS. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*. IEEE, 861–868.

Raman Arora, Poorya Mianjy, and Teodor Marinov. 2016. Stochastic optimization for multiview representation learning using partial least squares. In *International Conference on Machine Learning*. 1786–1794.

L. Balzano and S. J Wright. 2013. On GROUSE and incremental SVD. In *IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. IEEE, 1–4.

Christos Boutsidis, Dan Garber, Zohar Karnin, and Edo Liberty. 2015. Online principal components analysis. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 887–901.

Thierry Bouwmans and El Hadi Zahzah. 2014. Robust PCA via principal component pursuit: A review for a comparative evaluation in video surveillance. *Computer Vision and Image Understanding* 122 (2014), 22–34.

Kamalika Chaudhuri, Anand D Sarwate, and Kaushik Sinha. 2013. A near-optimal algorithm for differentially-private principal components. *The Journal of Machine Learning Research* 14, 1 (2013), 2905–2943.

Amey Desai, Mina Ghashami, and Jeff M Phillips. 2016. Improved practical matrix sketching with guarantees. *IEEE Transactions on Knowledge and Data Engineering* 28, 7 (2016), 1678–1690.

Amol Deshpande, Carlos Guestrin, Samuel R Madden, Joseph M Hellerstein, and Wei Hong. 2004. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 588–599.

C. Eckart and G. Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika* 1 (1936), 211–218. `https://doi.org/10.1007/BF02288367`

Armin Eftekhari, Raphael A Hauser, and Andreas Grammenos. 2018. MOSES: A Streaming Algorithm for Linear Dimensionality Reduction. *arXiv preprint arXiv:1806.01304* (2018).

Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).

Mina Ghashami, Edo Liberty, Jeff M Phillips, and David P Woodruff. 2016. Frequent directions: Simple and deterministic matrix sketching. *SIAM J. Comput.* 45, 5 (2016), 1762–1792.

Mina Ghashami and Jeff M Phillips. 2014. Relative errors for deterministic low-rank matrix approximations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 707–717.

Lie He, An Bian, and Martin Jaggi. 2018. Cola: Decentralized linear learning. In *Advances in Neural Information Processing Systems*. 4536–4546.

MA Iwen and BW Ong. 2016. A distributed and incremental svd algorithm for agglomerative data analysis on large networks. *SIAM J. Matrix Anal. Appl.* 37, 4 (2016), 1699–1718.

Ian Jolliffe. 2011. Principal component analysis. In *International encyclopedia of statistical science*. Springer, 1094–1096.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

Edo Liberty. 2013. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 581–588.

Luo Luo, Cheng Chen, Zhihua Zhang, Wu-Jun Li, and Tong Zhang. 2017. Robust Frequent Directions with Application in Online Learning. *arXiv preprint arXiv:1705.05067* (2017).

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

Teodor Vanislavov Marinov, Poorya Mianjy, and Raman Arora. 2018. Streaming Principal Component Analysis in Noisy Settings. In *International Conference on Machine Learning*. 3410–3419.

Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).

L. Mirsky. 1966. Symmetric gauge functions and unitarily invariant norms. *Quart. J. Math. Oxford* (1966), 1156–1159.

Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. 2013. Memory limited, streaming PCA. In *Advances in Neural Information Processing Systems*. 2886–2894.

I. Mitliagkas, C. Caramanis, and P. Jain. 2014. Streaming PCA with many missing entries. *Preprint* (2014).

Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. 2005. Streaming pattern discovery in multiple time-series. In *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 697–708.

Karl Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, 11 (1901), 559–572.

Radim Rehurek. 2011. Subspace tracking for latent semantic analysis. In *European Conference on Information Retrieval*. Springer, 289–300.

Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. 2017. Federated multi-task learning. In *Advances in Neural Information Processing Systems*. 4424–4434.

Andrew S Tanenbaum and Maarten Van Steen. 2007. *Distributed systems: principles and paradigms*. Prentice-Hall.

David P Woodruff et al. 2014. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science* 10, 1–2 (2014), 1–157.

Bin Yang. 1995. Projection approximation subspace tracking. *IEEE Transactions on Signal processing* 43, 1 (1995), 95–107.

A Supplementary Material

We start, by providing the full proof for Lemma 1.

Proof. Let the singular values of $\mathbf{Y}$ be the positive square root of the eigenvalues of $\mathbf{Y}\mathbf{Y}^T$; then by using the previously defined streaming block decomposition of a matrix $\mathbf{Y}$ we have the following,

$$\begin{aligned}\mathbf{Y}\mathbf{Y}^* &= \sum_{i=1}^{M} \mathbf{Y}^i(\mathbf{Y}^i)^* \\ &= \sum_{i=1}^{M} \hat{\mathbf{U}}^i\hat{\mathbf{\Sigma}}^i(\hat{\mathbf{V}}^i)^T(\hat{\mathbf{V}}^i)(\hat{\mathbf{\Sigma}}^i)^T(\hat{\mathbf{U}}^i)^T \\ &= \sum_{i=1}^{M} \hat{\mathbf{U}}^i\hat{\mathbf{\Sigma}}^i(\hat{\mathbf{\Sigma}}^i)^T(\mathbf{U}^i)^T\end{aligned}$$

Equivalently, the singular values of $\mathbf{Z}$ are similarly defined as the square root of the eigenvalues of $\mathbf{Z}\mathbf{Z}^T$.

$$\mathbf{Z}\mathbf{Z}^T = \sum_{i=1}^{M}(\hat{\mathbf{U}}^i\hat{\mathbf{\Sigma}}^i)(\hat{\mathbf{U}}^i\hat{\mathbf{\Sigma}}^i)^* = \sum_{i=1}^{M} \hat{\mathbf{U}}^i\hat{\mathbf{\Sigma}}^i(\hat{\mathbf{\Sigma}}^i)^*(\hat{\mathbf{U}}^i)^*$$

Thus $\mathbf{Y}\mathbf{Y}^T = \mathbf{Z}\mathbf{Z}^T$, hence the singular values of matrix $\mathbf{Z}$ must surely equal to those of matrix $\mathbf{Y}$. Moreover, since the left singular vectors of both $\mathbf{Y}$ and $\mathbf{Z}$ will be also eigenvectors of $\mathbf{Y}\mathbf{Y}^T$ and $\mathbf{Z}\mathbf{Z}^T$, respectively; then the eigenspaces associated with each - possibly repeated - eigenvalue will also be equal thus $\hat{\mathbf{U}} = \hat{\mathbf{U}}'\mathbf{B}$. The block diagonal unitary matrix $\mathbf{B}$ which has b unitary blocks of size $b \times b$ for each repeated eigenvalue; this enables the singular vectors which are associated with each repeated singular value to be rotated in the desired matrix representation $\hat{\mathbf{U}}$. □

We continue with the proof of Lemma 2, which begins by proving Lemma 2.1

Proof. If $\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is the SVD of $\mathbf{Y}$, then $\mathbf{Y}\mathbf{P} = \mathbf{U}\mathbf{\Sigma}\left(\mathbf{V}^T\mathbf{P}\right)$. Since $\mathbf{V}' = \mathbf{P}^T\mathbf{V}$ is orthogonal, $\mathbf{U}\mathbf{\Sigma}(\mathbf{V}')^T$ is the SVD of $\mathbf{Y}\mathbf{P}$. Hence, both $\mathbf{Y}$ and $\mathbf{Y}\mathbf{P}$ have the same singular values and left principal subspaces. □

We then finalise our proof of Lemma 2 by proving Lemma 2.2

Proof. At iteration $k \in [K]$, SPCA computes $\hat{\mathbf{Y}}^{\mathrm{SPCA}}_{[kb],r}$, the best rank-$r$ approximation of $\mathbf{Y}_{[kb]}$ using iteration (5). Hence, for each $k \in [K]$, the error of the approximation is given by $\|\mathbf{Y}_{[kb]} - \hat{\mathbf{Y}}^{\mathrm{SPCA}}_{[kb]}\|_F = \rho_r(\mathbf{Y}_{[kb]})$. Let $r_{\min} = r_{\min}(\alpha,\beta)$ and $r_{\max} = r_{\max}(\alpha,\beta) > 0$ be the minimum and maximum rank estimates in when running SAPCA. The result follows from

$$\rho_{r_{\max}(\alpha,\beta)}(\mathbf{Y}_{[kb]}) \leq \|\mathbf{Y}_{[kb]} - \hat{\mathbf{Y}}^{\mathrm{SAPCA}}_{[kb]}\|_F \leq \rho_{r_{\min}(\alpha,\beta)}(\mathbf{Y}_{[kb]}).$$

□

Now we present in Algorithm 4 the full implementation of the basic subspace merging algorithm which is a direct consequence of Lemma 1, with the only addition of the forgetting factor λ that only serves the purpose of giving more significance to the "newer" subspace. Following this, we can improve upon Algorithm 4 by exploiting the fact that the input subspaces are already *orthonormal* hence we can transform the Algorithm 4 to Algorithm 5. The key intuition comes from the fact that we can incrementally update $\mathbf{U}$ by using $\mathbf{U} \leftarrow \mathbf{Q}_p\mathbf{U}_R$. To do this we need to first create a subspace basis which spans $\mathbf{U_1}$ and $\mathbf{U_2}$, namely $\mathrm{span}(\mathbf{Q}_p) = \mathrm{span}([\mathbf{U_1}, \mathbf{U_2}])$. This is done by performing $[\mathbf{Q}_p, \mathbf{R}_p] = \mathrm{QR}([\lambda_1\mathbf{U}_1\mathbf{\Sigma}_1, \lambda_2\mathbf{U}_2\mathbf{\Sigma}_2])$ and use $\mathbf{R}_p$ to perform an incremental update. However, $\mathbf{R}_p$ is not diagonal, so a further SVD needs to be applied on it, which yields the required singular values and the rotation matrix to apply onto $\mathbf{Q}_p$ to represent the new subspace basis.

Algorithm 4: Basic MergeSubspaces algorithm

Data: $\mathbf{U}_1 \in \mathbb{R}^{d\times r_1}$, first subspace
$\mathbf{\Sigma}_1 \in \mathbb{R}^{r_1\times r_1}$, first subspace singular values
$\mathbf{U}_2 \in \mathbb{R}^{d\times r_2}$, second subspace
$\mathbf{\Sigma}_2 \in \mathbb{R}^{r_2\times r_2}$, second subspace singular values
$r \in [r]$, , the desired rank r
$\lambda \in (0,1)$, forgetting factor
$\lambda_2 \geq 1$, enhancing factor
Result: $\mathbf{U}' \in \mathbb{R}^{d\times r}$, merged subspace
$\mathbf{\Sigma}' \in \mathbb{R}^{r\times r}$, merged singular values
$[\mathbf{U}', \mathbf{\Sigma}', \tilde{}] \leftarrow \mathrm{SVD}_r([\lambda_1\mathbf{U}_1\mathbf{\Sigma}_1, \lambda_2\mathbf{U}_2\mathbf{\Sigma}_2])$

Algorithm 5: Faster MergeSubspaces algorithm

Data: $U_1 \in \mathbb{R}^{d\times r_1}$, first subspace
$\mathbf{\Sigma}_1 \in \mathbb{R}^{r_1\times r_1}$, first subspace singular values
$\mathbf{U}_2 \in \mathbb{R}^{d\times r_2}$, second subspace
$\mathbf{\Sigma}_2 \in \mathbb{R}^{r_2\times r_2}$, second subspace singular values
$r \in [r]$, , the desired rank r
$\lambda_1 \in (0,1)$, forgetting factor
$\lambda_2 \geq 1$, enhancing factor
Result: $\mathbf{U}' \in \mathbb{R}^{d\times r}$, merged subspace
$\mathbf{\Sigma}' \in \mathbb{R}^{r\times r}$, merged singular values
$[\mathbf{Q}_p, \mathbf{R}_p] \leftarrow \mathrm{QR}(\lambda_1\mathbf{U}_1\mathbf{\Sigma}_1 \mid \lambda_2\mathbf{U}_2\mathbf{\Sigma}_2)$
$[\mathbf{U}_R, \mathbf{\Sigma}', \tilde{}] \leftarrow \mathrm{SVD}_r(\mathbf{R_p})$
$\mathbf{U}' \leftarrow \mathbf{Q}_p\mathbf{U}_R$

B SAPCA Inherited Guarantees

We note that SAPCA in Algorithm 3 inherits some theoretical guarantees from MOSES presented in Eftekhari et al. (2018). Specifically, let μ be an *unknown* probability measure supported on $\mathbb{R}^d$ with zero mean. The informal objective is to find an r-dimensional subspace $\mathcal{U}$ that provides the best approximation with respect to the mass of μ. That is, provided that y is drawn from μ, the target is to find an r-dimensional subspace $\mathcal{U}$ that minimises the *population risk* by solving

$$\min_{\mathcal{U}\in\mathbb{G}(d,r)} \mathop{\mathbb{E}}_{\mathbf{y}\sim\mu} \|\mathbf{y} - \mathbf{P}_{\mathcal{U}}\mathbf{y}\|_2^2 . \tag{7}$$

where the Grassmanian $\mathbb{G}(d,r)$ is the manifold of all r-dimensional subspaces in $\mathbb{R}^d$ and $\mathbf{P}_{\mathcal{U}} \in \mathbb{R}^{d\times d}$ is the orthogonal projection onto $\mathcal{U}$. Unfortunately, the value of μ is unknown and cannot be used to directly solve (7), but provided we have access to a block of samples $\{\mathbf{y}_t\}_{t=1}^{\tau} \in \mathbb{R}^d$ that are independently drawn from μ, then (7) can be reformulated using the *empirical risk* by

$$\min_{\mathcal{U}\in\mathbb{G}(d,r)} \frac{1}{\tau}\sum_{t=1}^{\tau} \|\mathbf{y}_t - \mathbf{P}_{\mathcal{U}}\mathbf{y}_t\|_2^2 . \tag{8}$$

Given that $\sum_{t=1}^{\tau} \|\mathbf{y}_t - \mathbf{P}_{\mathcal{U}}\mathbf{y}_t\|_2^2 = \|\mathbf{Y}_\tau - \mathbf{P}_{\mathcal{U}}\mathbf{Y}_\tau\|_F^2$, it follows by the EYM Theorem Eckart and Young (1936); Mirsky (1966), that $\mathbf{P}_{\mathcal{U}}\mathbf{Y}_\tau$ is the *best* rank-r approximation to $\mathbf{Y}_\tau$ which is given by $\mathbf{Y}_{\tau,r} = \mathrm{SVD}_r(\mathbf{Y}_\tau)$. Therefore, $\mathcal{U} = \mathcal{Y}_{\tau,r} = \mathrm{span}(\mathbf{Y}_{\tau,r})$, which implies that $\|\mathbf{Y}_\tau - \mathbf{P}_{\mathcal{Y}_{\tau,r}}\mathbf{Y}_\tau\|_F^2 = \|\mathbf{Y}_\tau - \mathbf{Y}_{\tau,r}\|_F^2 = \rho_r^2(\mathbf{Y}_\tau)$, so the solution of (8) equals $\rho_r^2(\mathbf{Y}_\tau)/\tau$. For completeness the full MOSES theorem is shown below.

Theorem 1 (Moses Eftekhari et al. (2018))**.** *Suppose $\{\mathbf{y}_t\}_{t=1}^{\tau} \subset \mathbb{R}^d$ are independently drawn from a zero-mean Gaussian distribution with covariance matrix $\mathbf{\Xi} \in \mathbb{R}^{d\times d}$ and form $\mathbf{Y}_\tau = [\mathbf{y}_1 \cdots \mathbf{y}_\tau] \in \mathbb{R}^{d\times\tau}$. Let $\lambda_1 \geq \cdots \geq \lambda_d$ be the eigenvalues of $\mathbf{\Xi}$ and $\rho_r^2 = \rho_r^2(\Xi)$ be its residual. Define*

$$\eta_r = \frac{\lambda_1}{\lambda_r} + \sqrt{\frac{2\alpha\rho_r^2}{p^{\frac{1}{3}}\lambda_r}}, \tag{9}$$

Let $\widehat{\mathbf{Y}}_{\tau,r}$ be defined as in (5)*, $\widehat{\mathcal{Y}}_{\tau,r} = \mathrm{span}(\widehat{\mathbf{Y}}_{\tau,r})$ and α, p, c be constants such that $1 \leq \alpha \leq \sqrt{\tau/\log\tau}$, $p > 1$ and $c > 0$. Then, if $b \geq \max(\alpha p^{\frac{1}{3}} r (p^{\frac{1}{6}} - 1)^{-2}, c\alpha r)$ and $\tau \geq p\eta_r^2 b$, it holds, with probability at most $\tau^{-c\alpha^2} + e^{-c\alpha r}$ that*

$$\frac{\|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F^2}{\tau} \lesssim G_{\alpha,b,p,r,\tau} \tag{10}$$

$$\mathop{\mathbb{E}}_{\mathbf{y}\sim\mu} \|\mathbf{y} - \mathbf{P}_{\widehat{\mathcal{Y}}_{\tau,r}}\mathbf{y}\|_2^2 \lesssim G_{\alpha,b,p,r,\tau} + \alpha(d-r)\lambda_1\sqrt{\frac{\log\tau}{\tau}} \tag{11}$$

where

$$G_{\alpha,b,p,r,\tau} = \frac{\alpha p^{\frac{1}{3}} 4^{p\eta_r^2}}{(p^{\frac{1}{3}} - 1)^2} \min\left(\frac{\lambda_1}{\lambda_r}\rho_r^2, r\lambda_1 + \rho_r^2\right) \left(\frac{\tau}{p\eta_r^2 b}\right)^{p\eta_r^2 - 1}.$$

Notably, the condition of $\tau \geq p\eta_r^2 b$ is only required in order to obtain a tidy bound and is not necessary in the general case. Moreover, this implies that, when considering only asymptotic dominant terms of Theorem 1,

$$\|\mathbf{Y}_\tau - \widehat{\mathbf{Y}}_{\tau,r}\|_F^2 \propto \left(\frac{\tau}{b}\right)^{p\eta_r^2 - 1} \|\mathbf{Y}_\tau - \mathbf{Y}_{\tau,r}\|_F^2, \tag{12}$$

Practically speaking, assuming $\mathrm{rank}(\boldsymbol{\Xi}) \leq r$ and $\rho_r^2(\Xi) = \sum_{i=r+1}^d \lambda_i(\boldsymbol{\Xi})$ we can read that, $\widehat{\mathbf{Y}}_{\tau,r} = \mathbf{Y}_{\tau,r} = \mathbf{Y}_\tau$ meaning that the outputs of offline truncated SVD and Eftekhari et al. (2018) coincide.

B.1 Interpretation of SPCA as streaming, stochastic algorithm for PCA

It is easy to interpret SPCA as a streaming, stochastic algorithm for PCA. To see this, note that (7) is equivalent to maximising $\mathbb{E}_{y\sim\mu}\|\mathbf{U}\mathbf{U}^T\mathbf{y}\|_F^2$ over $\mathcal{Z} = \{\mathbf{U} \in \mathbb{R}^{d\times r} : \mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r}\}$ The restriction $\mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r}$ can be relaxed to $\mathbf{U}^*\mathbf{U} \preccurlyeq \mathbf{I}_r$, where $\mathbf{A} \preccurlyeq \mathbf{B}$ denotes that $\mathbf{B} - \mathbf{A}$ is a positive semi-definite matrix. Using the Schur's complement, we can formulate this program to

$$\begin{aligned} \max \quad & \mathop{\mathbb{E}}_{y\sim\mu} \langle \mathbf{U}\mathbf{U}^T, \mathbf{y}\mathbf{y}^T \rangle \\ \text{s. t.} \quad & \begin{bmatrix} \mathbf{I}_n & \mathbf{U} \\ \mathbf{U}^T & \mathbf{I}_r \end{bmatrix} \succcurlyeq \mathbf{0} \end{aligned} \tag{13}$$

Note that, (13) has an objective function that is convex and that the feasible set is also conic and convex. However, its gradient can only be computed when μ, since otherwise $\boldsymbol{\Xi} = \mathbb{E}[\mathbf{y}\mathbf{y}^T] \in \mathbb{R}^{d\times d}$ is unknown. If μ is known, and an iterate of the form $\widehat{\mathbf{S}}_t$ is provided, we could draw a random vector $\mathbf{y}_{t+1} \in R^d$ moving along the direction of $2\mathbf{y}_{t+1}\mathbf{y}_{t+1}^*\widehat{\mathbf{S}}_t$. This is because $\mathbb{E}[2\mathbf{y}_{t+1}\mathbf{y}_{t+1}^*\widehat{\mathbf{S}}_t] = 2\boldsymbol{\Xi}\widehat{\mathbf{S}}_t$ which is then followed by back-projection onto the feasible set $\mathcal{Z}$. Namely,

$$\widehat{\mathbf{S}}_{t+1} = \mathcal{P}\left(\mathbf{S}_t + 2\alpha_{t+1}\mathbf{y}_{t+1}\mathbf{y}_{t+1}^*\widehat{\mathbf{S}}_t\right), \tag{14}$$

One can see that in (14), $\mathcal{P}(\mathbf{A})$ projects onto the unitary ball of the spectral norm by clipping at one all of $\mathbf{A}$'s singular values exceeding one.

C Evaluation Details

C.1 Synthetic Datasets

For the tests on synthetic datasets, the vectors $\{\mathbf{y}_t\}_{t=1}^\tau$ are drawn independently from a zero-mean Gaussian distribution with the covariance matrix $\boldsymbol{\Xi} = \mathbf{S}\boldsymbol{\Lambda}\mathbf{S}^T$, where $\mathbf{S} \in \mathrm{O}(d)$ is a generic basis obtained by orthogonalising a standard random Gaussian matrix. The entries of the diagonal matrix $\boldsymbol{\Lambda} \in \mathbb{R}^{d\times d}$ (the eigenvalues of the covariance matrix $\boldsymbol{\Xi}$) are selected according to the power law, namely, $\lambda_i = i^{-\alpha}$, for a positive α. To be more succinct, wherever possible we employ MATLAB's notation for specifying the value ranges in this section.

To assess the performance of SAPCA, we let $\mathbf{Y}_t = [\mathbf{y}_1, \cdots, \mathbf{y}_t] \in \mathbb{R}^{d \times t}$ be the data received by time t and $\widehat{\mathbf{Y}}_{t,r}^{\text{spca}}$ be the output of SPCA at time t. [4] Then, the error incurred by SAPCA is

$$\frac{1}{t}\|\mathbf{Y}_t - \widehat{\mathbf{Y}}_{t,r}^{\text{sapca}}\|_F^2, \tag{15}$$

Recall, that the above error is always larger than the residual of $\mathbf{Y}_t$, namely,

$$\|\mathbf{Y}_t - \widehat{\mathbf{Y}}_{t,r}^{\text{sapca}}\|_F^2 \geq \|\mathbf{Y}_t - \mathbf{Y}_{t,r}\|_F^2 = \rho_r^2(\mathbf{Y}_t). \tag{16}$$

In the expression above, $\mathbf{Y}_{t,r} = \text{SVD}_r(\mathbf{Y}_t)$ is a rank-r truncated SVD of $\mathbf{Y}_t$ and $\rho_r^2(\mathbf{Y}_t)$ is the corresponding residual.

Additionally, we compare SAPCA against SPCA, GROUSE Balzano and Wright (2013), FD Desai et al. (2016), PM Mitliagkas et al. (2013) and a version of PAST Papadimitriou et al. (2005); Yang (1995). Interestingly and contrary to both SPCA & SAPCA, the aforementioned algorithms are *only* able to estimate the principal components of the data and *not* their projected data on-the-fly. Although, it has to noted that in this setup we are only interested in the resulting subspace $\mathcal{U}$ along with its singular values Σ but is worth mentioning that the projected data, if desired, can be kept as well. More specifically, let $\widehat{\mathcal{S}}_{t,r}^{g} \in \text{G}(d, r)$ be the span of the output of GROUSE, with the outputs of the other algorithms defined similarly. Then, these algorithms incur in errors

$$\frac{1}{t}\|\mathbf{Y}_t - \mathbf{P}_{\widehat{\mathcal{S}}_{t,r}^{v}}\mathbf{Y}_t\|_F^2, \;\; v \in g, f, p, \text{spca},$$

where we have used the notation $\mathbf{P}_{\mathcal{A}} \in \mathbb{R}^{d \times d}$ to denote the orthogonal projection onto the subspace $\mathcal{A}$. Even though robust FD Luo et al. (2017) improves over FD in the quality of matrix sketching, since the subspaces produced by FD and robust FD coincide, there is no need here for computing a separate error for robust FD.

Throughout our synthetic dataset experiments we have used an ambient dimension $d = 400$, and for each $a \in (0.0001, 0.001, 0.5, 1, 2, 3)$ generated $N = 4000$ feature vectors in $\mathbb{R}^d$ using the method above. This results in a set of with four datasets of size $\mathbb{R}^{d \times N}$. Furthermore, in our experiments we used a block size of $b = 50$ for SAPCA and SPCA, while for PM we chose $b = d$. FD & GROUSE perform singular updates and do not need a block-size value. Additionally, the step size for GROUSE was set to 2 and the total sketch size for FD was set $2r$. In all cases, unless otherwise noted in the respective graphs the starting rank for all methods in the synthetic dataset experiments was set to $r = 10$.

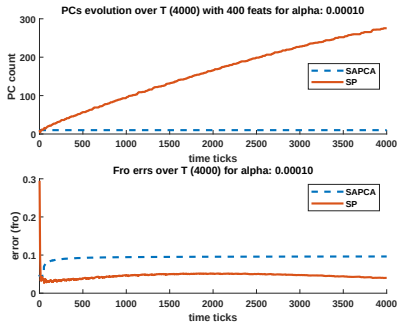
We evaluated our algorithm using the aforementioned error metrics on a set of datasets generated as described above. The results for the different a values are shown in Figure 3, which shows SAPCA can achieve an error that is significantly smaller than SP while maintaining a small number of principal components throughout the evolution of the algorithms in the absence of a forgetting factor λ. When a forgetting factor is used, as is shown in 4 then the performance of the two methods is similar. This figure was produced on pathological datasets generated with an adversarial spectrum. It can be seen that in SPIRIT the need for PC's increases dramatically for no apparent reason, whereas SAPCA behaves favourably.

Additionally, in order to bound our SAPCA algorithm in terms of the expected error, we used the SPCA with a low and high bound each set to the lowest and highest estimated r-rank of SAPCA during its execution. We fully expect SAPCA to fall within these bounds. On the other hand, Figure 4 shows that a drastic performance improvement occurs when using an exponential forgetting factor for SPIRIT with value $\lambda = 0.9$, but the generated subspace is of inferior quality when compared to the one produced by SAPCA.

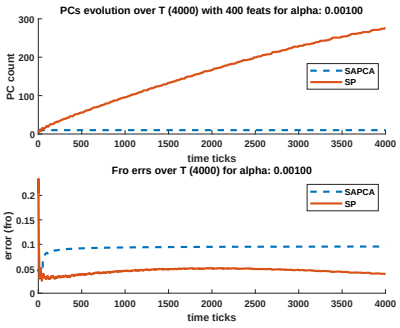
C.2 Real Datasets

Again as with the synthetic datasets, across all real dataset experiments we used an ambient dimension d and N equal to the dimensions of each dataset. Similarly to the previous section, we used a block

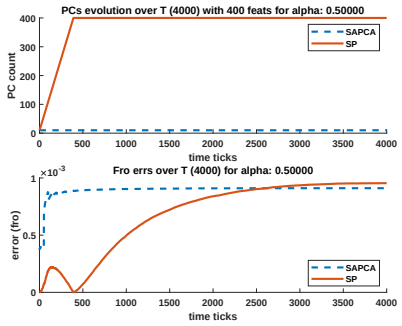
[4] Recall, since *block*-based algorithms like SAPCA, do not update their estimate after receiving feature vector but per each block for convenience in with respect to the evaluation against other algorithms (which might have different block sizes or singular updates), we properly *interpolate* their outputs over time.



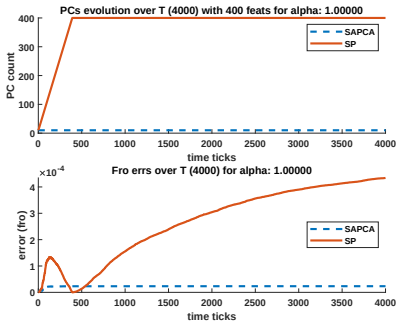
(a) $\alpha = 00.0001$.



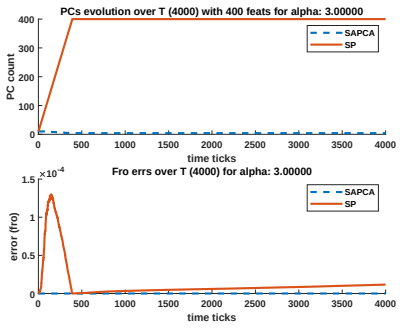
(b) $\alpha = 0.001$.



(c) $\alpha = 0.5$.



(d) $\alpha = 1$.

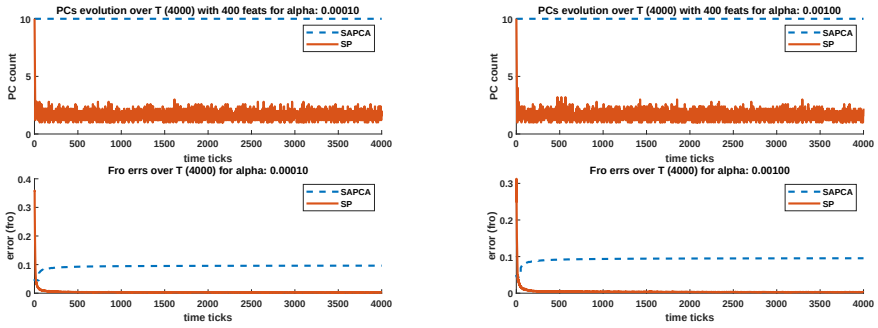


(e) $\alpha = 3$.

Figure 3: More Pathological examples for adversarial Spectrums.

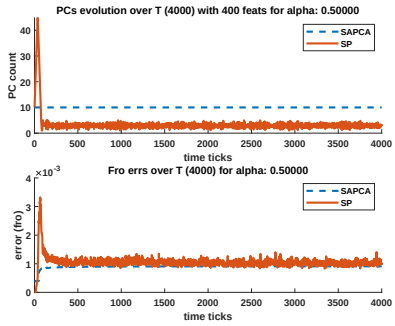
size of $b = 50$ for SAPCA, SPCA and $b = d$ for PM. The step size for GROUSE was again set to 2 and the total sketch size for FD equal to $2r$. Additionally, we used the same bounding technique as with the synthetic datasets to bound the error of SAPCA using SPCA with lowest and highest estimation of the r-rank and note that we fully expect SAPCA to fall again within these bounds. Finally, to enhance readability we split the errors over time in two plots SP's & GROUSE's exploding errors significantly skew the plots.

Humidity readings sensor node dataset evaluation. Firstly, we evaluate our methods against a dataset that has an ambient dimension $d = 48$ and is comprised out of $N = 7712$ total feature vectors thus its total size being $\mathbb{R}^{48\times 7712}$. This dataset is highly periodic in nature and has a larger lower/higher value deltas when compared to the other datasets. The initial rank used for all algorithms was $r = 10$. The errors are plotted in logarithmic scale and can be seen in Figure 1c and we can clearly see that SAPCA outperforms the competing algorithms while being within the expected SPCA_{low} & $\text{SPCA}_{\text{high}}$ bounds.

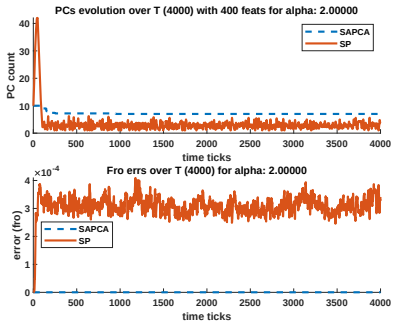


(a) $\alpha = 0.001$.

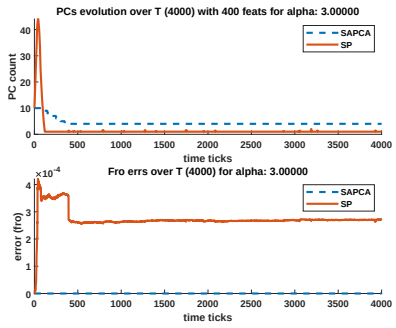
(b) $\alpha = 0.001$.



(c) $\alpha = 0.5$.



(d) $\alpha = 2$.

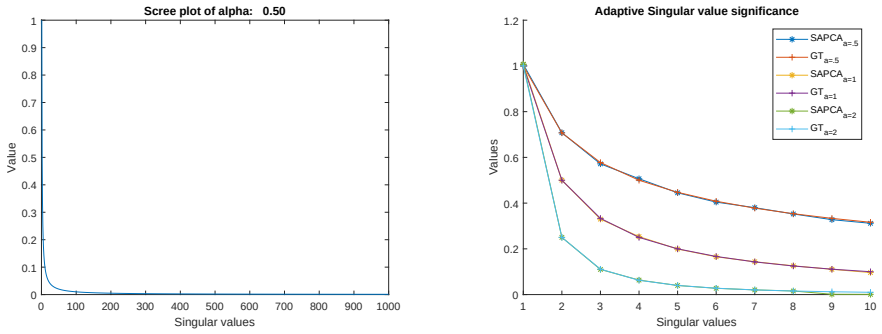


(e) $\alpha = 3$.

Figure 4: Performance measurements across the spectrum (when using forgetting factor $\lambda = 0.9$).

Light readings sensor node dataset evaluation. Secondly, we evaluate the aforementioned methods again a dataset that has an ambient dimension $d = 48$ and is comprised out of $N = 7712$ feature vectors thus making its total size $\mathbb{R}^{48\times7712}$. It contains mote light readings can be characterised as a much more volatile dataset when compared to the Humidity one as it contains much more frequent and rapid value changes while also having the highest value delta of all mote datasets evaluated. Again, as with Humidity dataset we used an initial seed rank $r = 10$ while keeping the rest of the parameters as described above, the errors over time for all algorithms is shown in Figure 1d plotted logarithmic scale. As before, SPCA outperforms the other algorithms while being again within the expected SPCA_{low} & $\text{SPCA}_{\text{high}}$ bounds.

Temperature readings sensor node dataset evaluation. The third dataset we evaluate contains temperature readings from the mote sensors and has an ambient dimension $d = 56$ containing $N = 7712$ feature vectors thus making its total size $\mathbb{R}^{56\times7712}$. Like the humidity dataset the temperature readings exhibit periodicity in their value change and rarely have spikes. As previously we used a seed rank of $r = 20$ and the rest of the parameters as described in the synthetic comparison above, the errors over time for all algorithms is shown in Figure 6a plotted in logarithmic scale. It

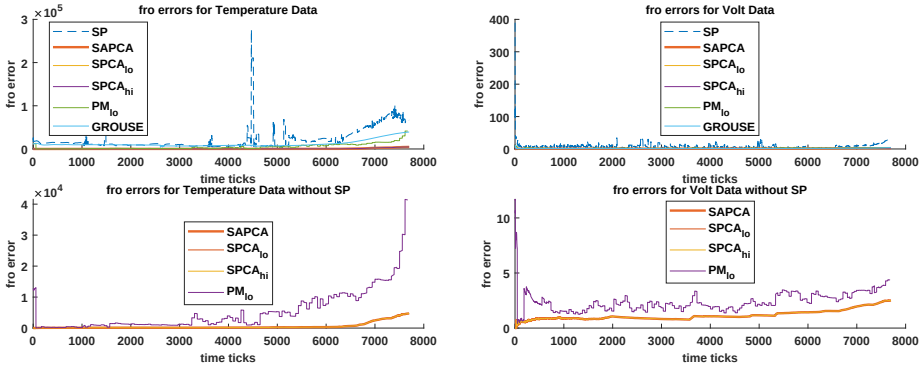


(a) Pathological spectrum ($\lambda = 1, \alpha = 0.5$). (b) Adaptive Singular Values estimation.

Figure 5: Pathological SV Spectrum Example (5a) and Adaptive SV estimation for various α's against the true ones (5b).

is again evident that SAPCA outperforms the other algorithms while being within the SPCA_{low} & $\text{SPCA}_{\text{high}}$ bounds.

Voltage readings sensor node dataset evaluation. Finally, the fourth and final dataset we evaluate has an ambient dimension of $d = 46$ contains $N = 7712$ feature vectors thus making its size $\mathbb{R}^{46\times 7712}$. Similar to the Light dataset this is an contains very frequent value changes, has large value delta which can be expected during operation of the nodes due to various reasons (one being duty cycling). As with the previous datasets we use a seed rank of $r = 10$ and leave the rest of the parameters as described previously. Finally, the errors over time for all algorithms is shown in Figure 6a and are plotted in logarithmic scale. As expected, SAPCA here outperforms the competing algorithms while being within the required error bounds.



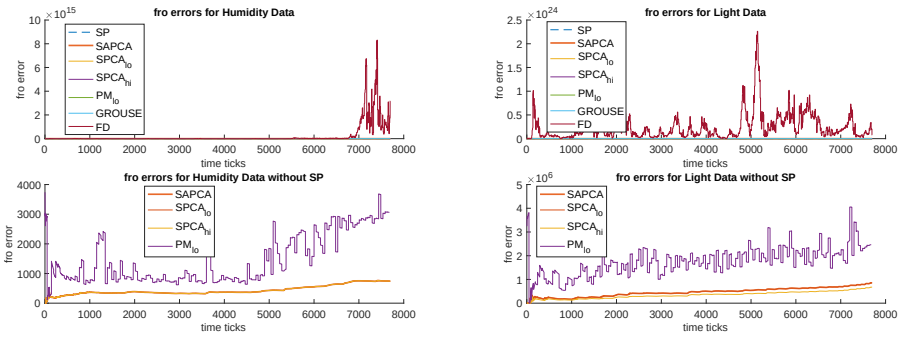
(a) Temperature. (b) Volt.

Figure 6: Comparisons against Temperature (6a) & Volt (6b) datasets with respect to the Frobenious norm error, Top figures include SPIRIT (SP), SAPCA, SPCA, PM, & GROUSE and Bottom figures only show SAPCA, SPCA, & PM due to exploding errors of the other methods.

C.3 Memory Evaluation

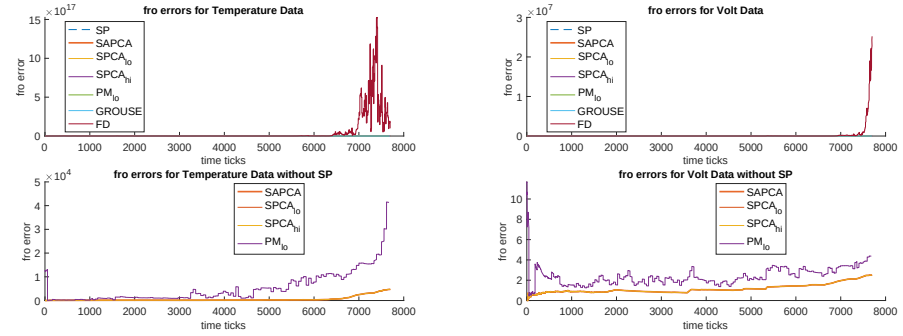
We benchmarked each of the methods used against its competitors and found our SAPCA algorithm to perform favourably. With respect to the experiments, in order to ensure accurate measurements, we started measuring after clearing the previous profiler contents. The tool used in all profiling instances was `MATLAB`'s built-in memory profiler.

These empirical results support the theoretical claims about the storage optimality of SAPCA. In terms of average and median memory allocations, SAPCA is most of the times better than the competitors. Naturally, since PM requires the materialisation of larger block sizes in turn it requires



(a) Humidity.

(b) Light.



(c) Temperature.

(d) Volt.

Figure 7: Performance for Real Datasets (with FD) is shown for Humidity (7a), Light (7b), Temperature (7c), and Volt (7d) respectively.

more memory than both SAPCA and FD. Moreover, GROUSE, in its reference implementation requires the instantiation of the whole matrix again; this is because the reference version of GROUSE is expected to run on a subset of a sparse matrix which is copied locally to the function - since in this instance we require the entirety of the matrix to be allocated and thus results in a large memory overhead. An improved, more efficient GROUSE implementation would likely solve this particular issue.

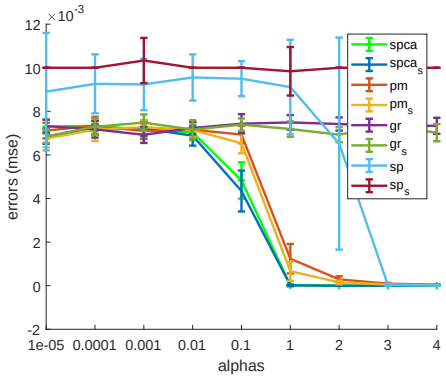
C.4 Federated Evaluation

As per our evaluation, our algorithm has the flexibility of not having its merges bounded by the ambient dimension d. This is especially true when operating on a memory limited scenario as the minimum number of feature vectors that need to be kept has to be a multiple of the ambient dimension d in order to provide their theoretical guarantees (such as in Mitliagkas et al. (2014)). Further, in the case of having an adversarial spectrum (e.g. as in Figure 5a), energy thresholding quickly overestimates the number of required principal components, unless a forgetting factor is used, but at the cost of approximation quality and robustness as it can me seen throughout our experiments when dealing with pathological spectrums. Notably, in a number of runs SP ended up with linearly dependent columns in the generated subspace and failed to complete. This is an inherent limitation of Gram-Schmidt orthonormalisation procedure used in the reference implementation and substituting it with a more robust one (such as QR) decreased its efficiency throughout our experiments.

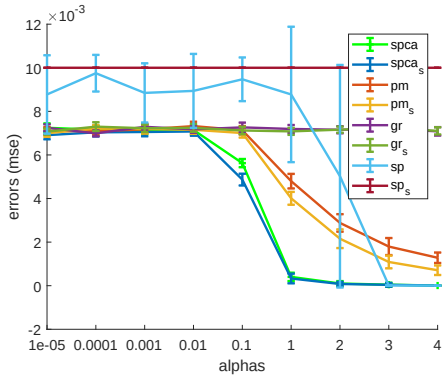
C.5 Time-Order Independence Empirical Evaluation

Apart from theoretically proving our time order independence hypothesis in Lemma 2, we also empirically validate it for a range of α values using $d = 100$ features and $n = 10$k. All algorithms were resilient - up to small errors - to $\mathbf{Y}$ permutation as was expected, but SAPCA being based heavily on SVD follows its produced subspaces much more evidently when compared against the other methods very closely. The figures show the errors for recovery ranks r 5 (8a), 20 (8b), 50 (8c),

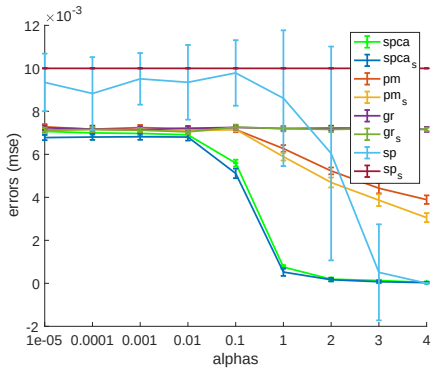
and 75 (8d). It has to be noted, that legends are subscripted with s (e.g. gr_s) compare against the SVD output while the others against its own output of the perturbation against the original $\mathbf{Y}$.



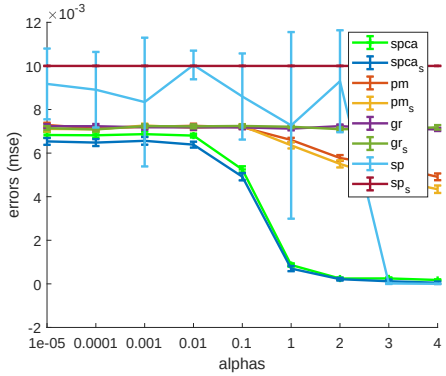
(a) Errors for $r = 5$.



(b) Errors for $r = 20$.



(c) Errors for $r = 50$.



(d) Errors for $r = 75$.

Figure 8: Mean Subspace errors of 20 permutations of $Y \in \mathbb{R}^{100\times 10000}$ for r equals 5 (a), 20 (b), 50 (c) and 75 (d).